Microprogramming Project

- You are to write and submit a microprogram to interpret the following 8-bit accumulator-based target machine instruction set (ISA) for a multicycle CPU design with a given datapath and microinstruction format using the MicroTiger microprogramming tool.
- MicroTiger is a graphical microcode simulator with a reconfigurable datapath that runs on any Windows PC.
- This tool was developed by Brian VanBuren for his CE MS Thesis at RIT entitled "Graphical Microcode Simulator with a Reconfigurable Datapath."
- From myCourses (or course web page) download the project package "550-project-winter2012.zip".
- The package contains an executable that runs under Windows, along with support files, start microprogram and ISA test programs. The project zipped file contains the following files:

  – **MicroTiger executable:** microtiger-student.exe

  – **Required program support DLLs:**

  mingwm10.dll wxbase26_gcc_custom.dll
  wxmsw26_core_gcc_custom.dll wxmsw26_html_gcc_custom.dll

  – **Target datapath file:** datapath.dp **.dp**

  – **Start microprogram file:** start-microprogram.mc **.mc**

  – **Five ISA test programs (will be used to evaluate the correctness of your microprogram):** **.isa**

  memory_inherent.isa memory_otherbranches.isa memory_otherslogical.isa
  memory_othersmath.isa memory_storebranch.isa

Due 12:00 Noon Monday, February 18

Microprogramming Project

- You do not have to install anything to run MicroTiger, just unzip all files to a single directory.
- **How To Use MicroTiger:** Once running MicroTiger, a full user guide is available through the Help->Contents menu.
- Modify/complete the given start microprogram to implement the project ISA. The given start microprogram initializes the datapath, fetches the instruction opcode byte and starts the decode process (each instruction, however, is just treated as a No-Op).
- For this project, submit your completed and fully-commented microprogram file (.mc) to the myCourses dropbox for the project.
- In addition to submitting your microprogram to the dropbox, you should submit **a written report** with the following items (electronic version to dropbox and printed copy):

(1) A brief description of your approach to the assignment, and the features of your solution.
(2) Dependent RTN statements for all the instructions implemented.
(3) The resulting CPI for the different instruction types.
(4) A statement of any relevant problems or difficulties encountered during the assignment.
(5) A listing of your fully-commented microprogram as submitted in dropbox..
(6) A description of testing procedures used and observations.
(7) Any additional remarks or conclusions you deem noteworthy of mention.

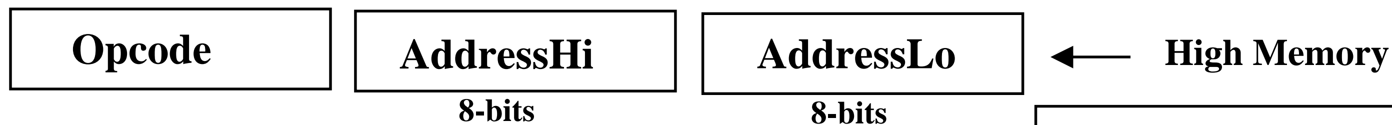
Target Instruction Set Architecture (ISA)

- **The target ISA is an 8-bit Accumulator-based ISA with variable length encoding (1-3 bytes).**
- **16-bit memory addressing:**
  - **Total addressable memory = 2^{16} = 64 KBytes = 65536 Bytes**
    (for program, data, stack space)
- **There are three types of instructions supported:**

(1) **Inherent instructions:** One-byte instructions, just the Opcode byte.

(2) **Immediate instructions**: Two-byte instructions. The instruction Opcode byte is followed by one additional byte of immediate data.

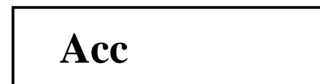
(3) **Memory reference instructions:** three-byte instructions. The instruction Opcode byte is followed by two bytes of address information. The high-order byte of the address appears in the byte memory location immediately following the Opcode byte (**Big Endian order**).



ISA Registers & Flags

- **16-bit Program Counter (PC)**

- **8-bit Accumulator**



- **16-bit Stack Pointer (SP)**

- **Four Flags or condition code bits NZVC**

  N = Negative Flag　　Z = Zero Flag

  V = Overflow Flag　　C = Carry Flag

- **The effect of an instruction on the flags is indicated by**
  "0" for flag is cleared (i.e. = 0)
  "1" for flag is set (i.e = 1)
  "-" for no change, and
  "x" for possible flag value change according to instruction result

Instruction Types:

Opcode

Inherent Instructions: One Byte Only

Opcode	Effect on Flags: NZVC
00000001 - **NOP** - No operation	- - - -
00010001 - **HALT** - Halt the machine	- - - -
00100001 - **CLA** - Clear the accumulator	0100
00110001 - **CMA** - 1's complement the accumulator	xx00
01000001 - **INCA** - Increment the accumulator by one	xx0x
01010001 - **DECA** - Decrement the accumulator by one	xx0x
01100001 - **RET** - Return from subroutine (post incrementing SP)	- - - -
01110001 - **ROLCA** - Circular shift Carry bit & Acc left 1 bit	xx0x
10000001 - **CLC** - Clear Carry Flag bit	- - - 0
10010001 - **STC** - Set Carry Flag bit	- - - 1

Effect of an instruction on the flags is indicated by:
- **"0" for flag is cleared (i.e. = 0)**
- **"1" for flag is set (i.e = 1)**
- **"-" for no change, and**
- **"x" for possible flag value change according to instruction result**

Instruction

Opcode

One Byte

Instruction Types:

Inherent Instructions Opcodes

Inherent Instructions Opcodes	Binary	Hex
NOP	00000001	01
HALT	00010001	11
CLA	00100001	21
CMA	00110001	31
INCA	01000001	41
DECA	01010001	51
RET	01100001	61
ROLCA	01110001	71
CLC	10000001	81
STC	10010001	91

X 1

Inherent Instructions Example:
ROLCA - Circular shift Carry bit & Acc left 1 bit

Opcode

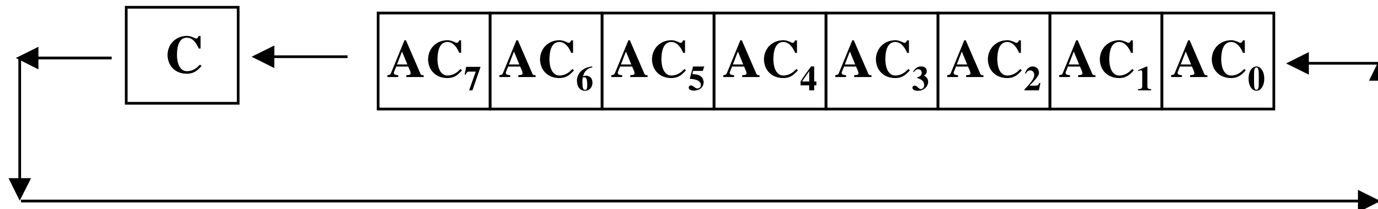
01110001 = 71 (Hex)

Independent RTN:

Instruction ← Mem[PC]
PC ← PC + 1
C ← AC_7 ; Accumulator ← $AC_6\, AC_5\, AC_4\, AC_3\, AC_1\, AC_1\, AC_o\, C$

Before ROLCA:

Carry Flag C **Accumulator**



After ROLCA:

Carry Flag C **Accumulator**

AC_7

Instruction Types:

Opcode +

Store and Branch Instructions: Two additional address bytes

Opcode	Effect on Flags: **NZVC**
00001z10 - **STA** - Store accumulator (use STAin for indirect)	- - - -
00011z10 - **JMP** - Unconditional branch **i.e. Jump**	- - - -
00101z10 - **JEQ** - Branch if equal to zero (Z=1)	- - - -
00111z10 - **JCS** - Branch if carry (C=1)	- - - -
01001z10 - **JLT** - branch if negative (N=1)	- - - -
01011z10 - **JVS** - branch if overflow (V=1)	- - - -
01101z10 - **JSR** - jump to subroutine (push PC on stack - predecrement SP)	- - - -

The z bit in the instruction Opcode determines the addressing mode used:

z = 0 Direct addressing, the address needed follows the Opcode

z = 1 Indirect addressing, the address following the Opcode is a pointer to the address needed by the instruction

Instruction

16 bit Address = AddressHi,Address Lo

Instruction Types:

Store and Branch Instructions Opcodes

Store and Branch Instructions Opcodes

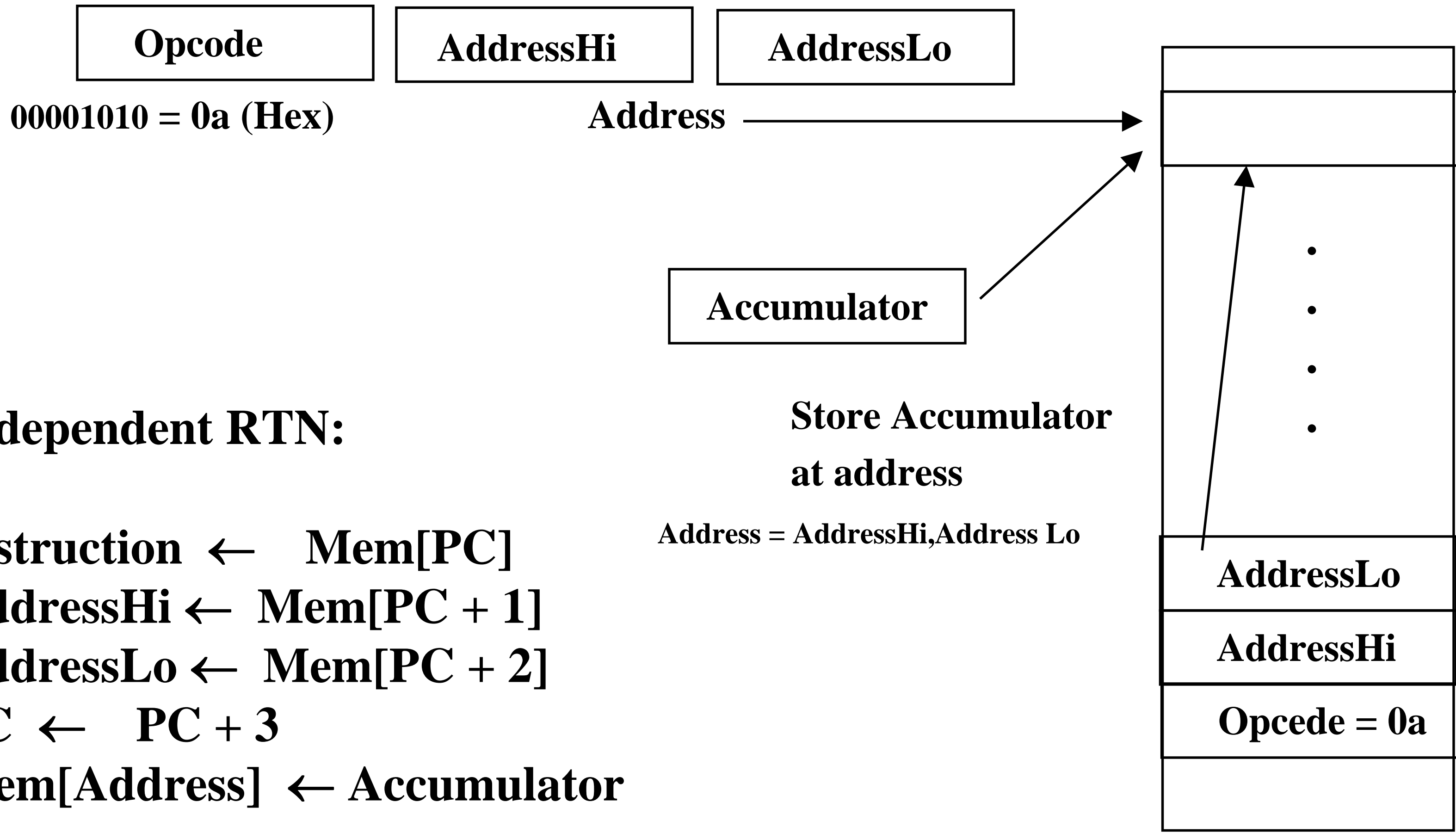
	Binary	Hex (Direct z=0)	Hex (Indirect z=1)
STA	00001z10	0a	0e
JMP	00011z10	1a	1e
JEQ	00101z10	2a	2e
JCS	00111z10	3a	3e
JLT	01001z10	4a	4e
JVS	01011z10	5a	5e
JSR	01101z10	6a	6e

Addressing Mode

X a

X e

Store and Branch Instructions Addressing Modes: Direct Addressing



Independent RTN:

Instruction ← Mem[PC]
AddressHi ← Mem[PC + 1]
AddressLo ← Mem[PC + 2]
PC ← PC + 3
Mem[Address] ← Accumulator

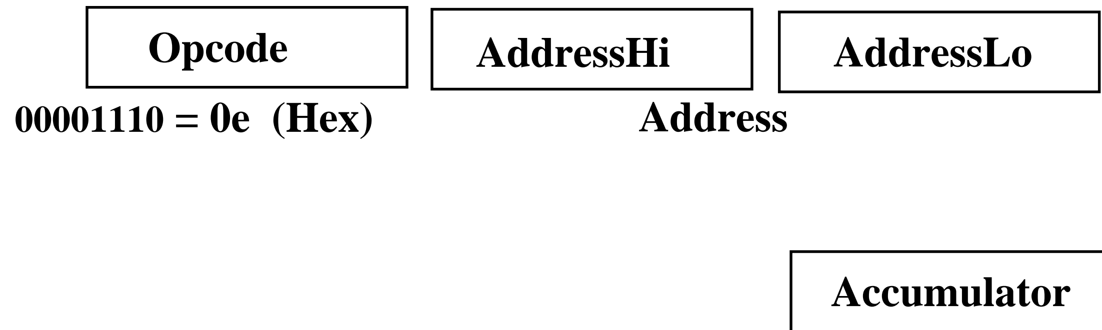
Address = AddressHi,Address Lo

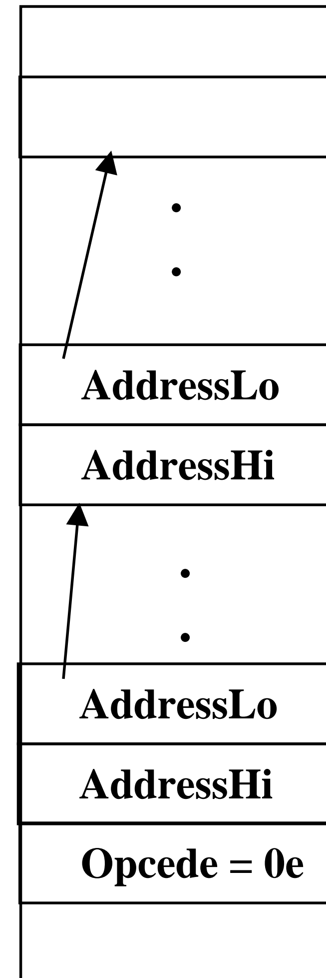
16 bit memory address – memory width one byte

Store and Branch Instructions Addressing Modes: Indirect Addressing

STA = Store Accumulator

Example Instruction STAind Store Accumulator Indirect (z=1 in Opcode)





Store Accumulator at indirect address

Address = AddressHi,Address Lo

Independent RTN:

Instruction ← Mem[PC]
AddressHi ← Mem[PC + 1]
AddressLo ← Mem[PC + 2]
PC ← PC + 3
Mem[Mem[Address]] ← Accumulator

Low Memory

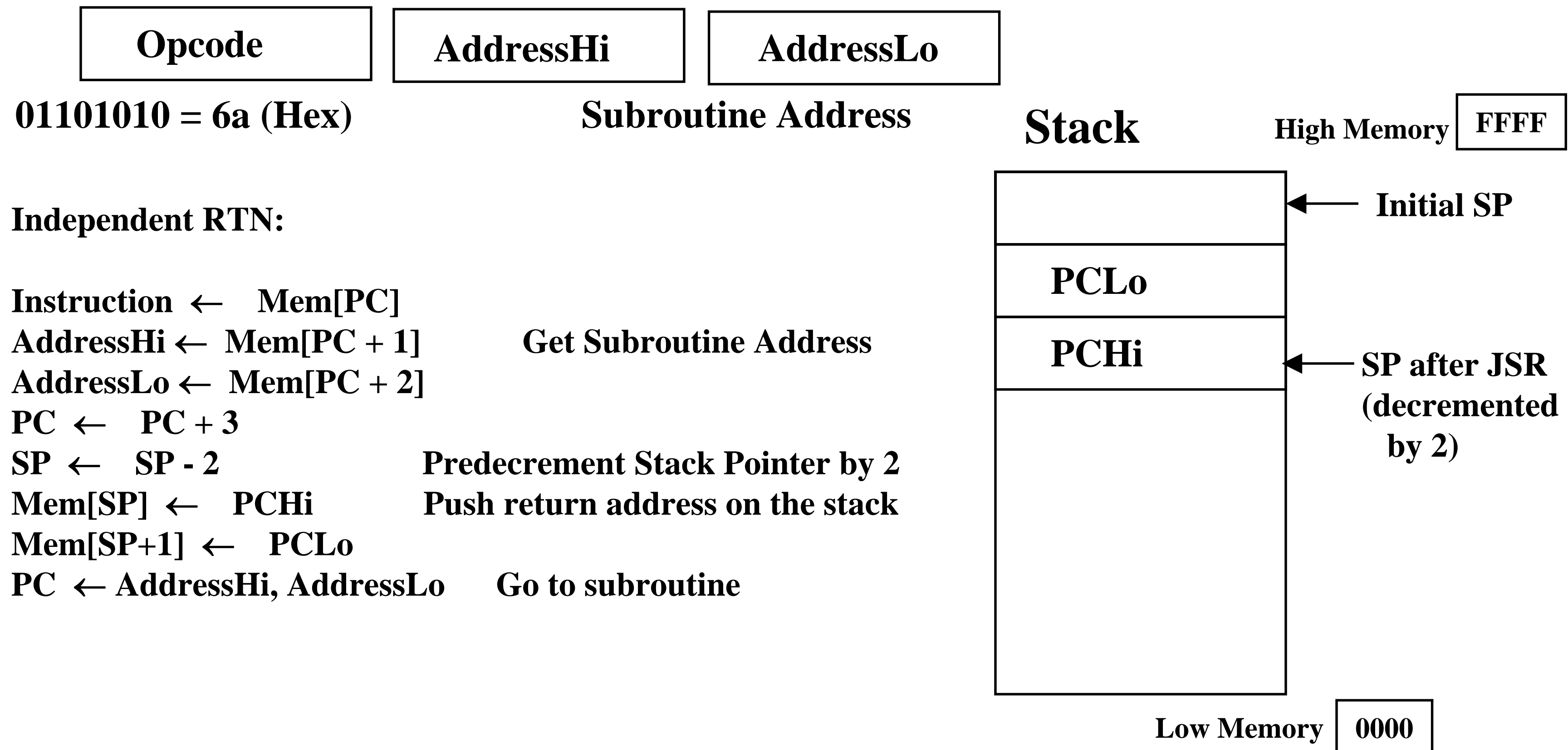
Address = AddressHi,Address Lo

16 bit memory address – memory width one byte

Subroutine Call/Return & Stack Operation

- **The Stack Pointer (SP) points to the last filled byte in the stack and is initialized in high memory** e.g initial SP = ffff
- **The stack grows to low memory as items are added**
- **JSR - Jump to Subroutine Direct** (push PC on stack - predecrement SP)

Instruction JSR Direct (z=0 in Opcode)



16 bit memory address – memory width one byte

Subroutine Call/Return & Stack Operation

RET - Return from Subroutine (post incrementing SP)

Inherent instruction one byte only – the Opcode

Instruction RET

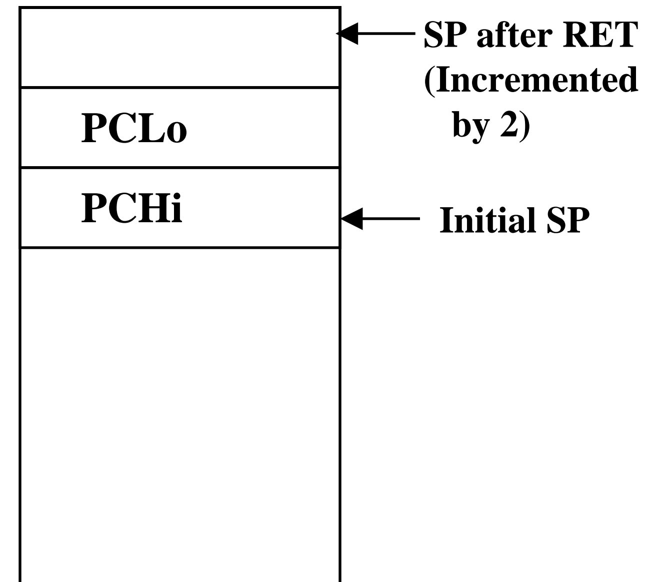
Opcode

01100001 = 61 (Hex)

Independent RTN:

Instruction ← Mem[PC]
PC ← PC + 1
PCHi ← Mem[SP] Get Return Address into PC
PCLo ← Mem[SP + 1]
SP ← SP + 2 Post increment Stack Pointer(SP) by 2

PC = PCHi,PCLo



16 bit memory address – memory width one byte

Instruction Types:

Opcode +

"Other" Instructions: One or Two Additional Bytes

Opcode	Effect on Flags: **NZVC**
0000yz11 - LDA - Load Acc (use LDAim for immediate)	xx00
0001yz11 - ORA - Inclusive OR to Accumulator	xx00
0010yz11 - EOR - Exclusive OR to Accumulator	xx00
0011yz11 - AND - Logical AND to Accumulator	xx00
0100yz11 - ADD - Add to Accumulator	xxxx
0101yz11 - SUBA - Subtract from Accumulator	xxxx
0110yz11 - LDS - Load stack pointer SP with 16-bit value	- - - -

The yz bits in the instruction Opcode determine the addressing mode used:

y = 0, z = 0 Direct addressing, the address of operand needed follows the Opcode

y = 1, z = 0 Immediate addressing, the one or two byte operands follows the Opcode

y = 0, z = 1 Indirect addressing, the address following the Opcode is a pointer to the address needed by the instruction

y =1, z = 1 Not allowed, illegal Opcode

Instruction Types:

“Other” Instructions Opcodes

Other Instructions Opcodes				
	Binary	Hex		
		Direct y = z =0	Indirect y=0 z =1	Immediate y=1 z=0
LDA	0000yz11	03	07	0b
ORA	0001yz11	13	17	1b
EOR	0010yz11	23	27	2b
AND	0011yz11	33	37	3b
ADD	0100yz11	43	47	4b
SUBA	0101yz11	53	57	5b
LDS	0110yz11	63	67	6b

Addressing Mode

X 3 X 7 X b

"Other" Instructions Addressing Modes: Immediate Addressing

LDA 0000yz11

LDA = Load Accumulator

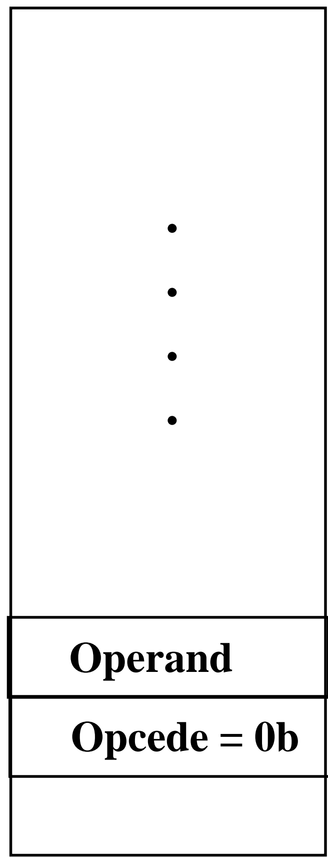
Example Instruction LDAImm Accumulator Immediate (y = 1 z = 0 in Opcode)

Opcode	Data Byte
00001011 = 0b (Hex)	Operand

i.e immediate operand byte to be loaded into Accumulator

Accumulator

Load Accumulator With immediate operand after opcode



Independent RTN:

Instruction ← Mem[PC]
Accumulator ← Mem[PC + 1]
PC ← PC + 2

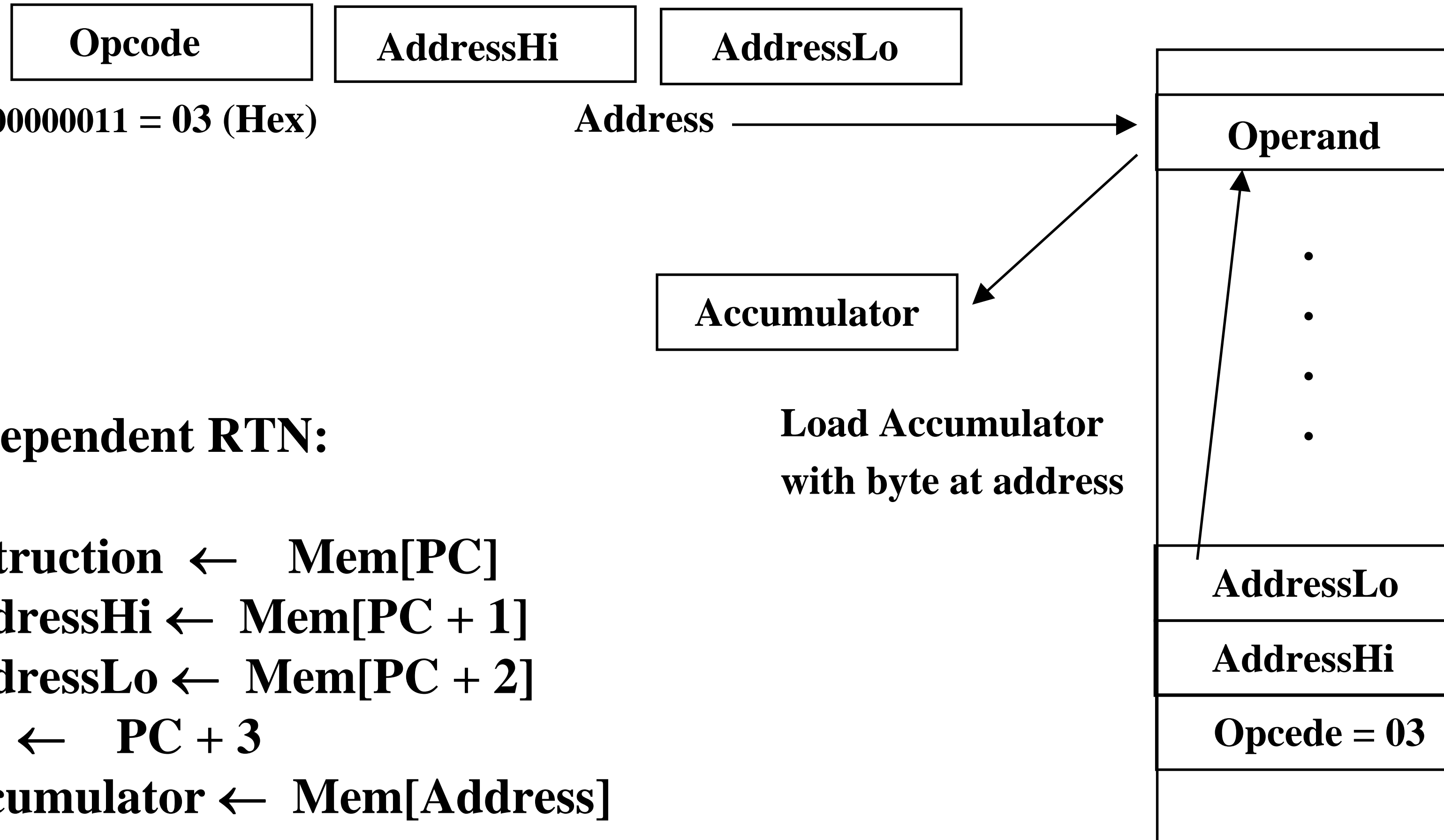
PC = PCHi,PCLo

16 bit memory address – memory width one byte

"Other" Instructions Addressing Modes: Direct Addressing

LDA 0000yz11

LDA = Load Accumulator



Low Memory

PC = PCHi,PCLo
Address = AddressHi,Address Lo

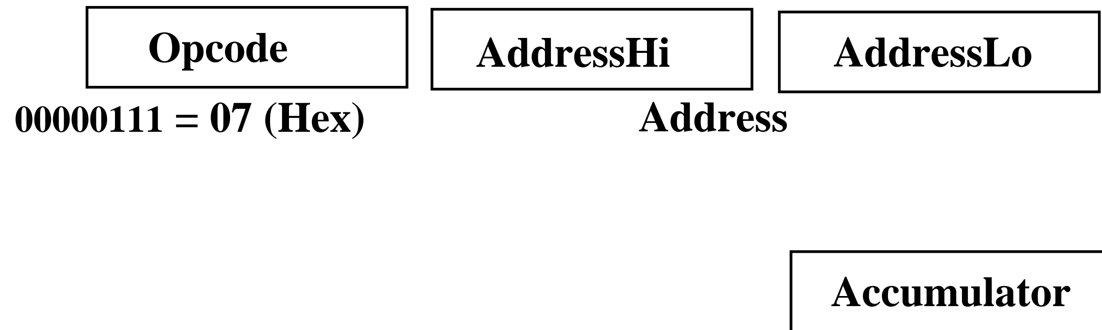
16 bit memory address – memory width one byte

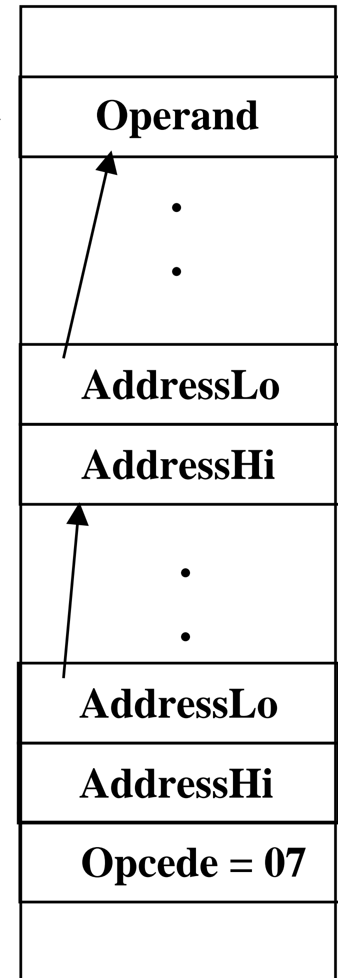
"Other" Instructions Addressing Modes: Indirect Addressing

LDA = Load Accumulator

LDA 0000yz11

Example Instruction LDAind Load Accumulator Indirect (y = 0 z = 1 in Opcode)





Load Accumulator with byte at indirect address

Low Memory

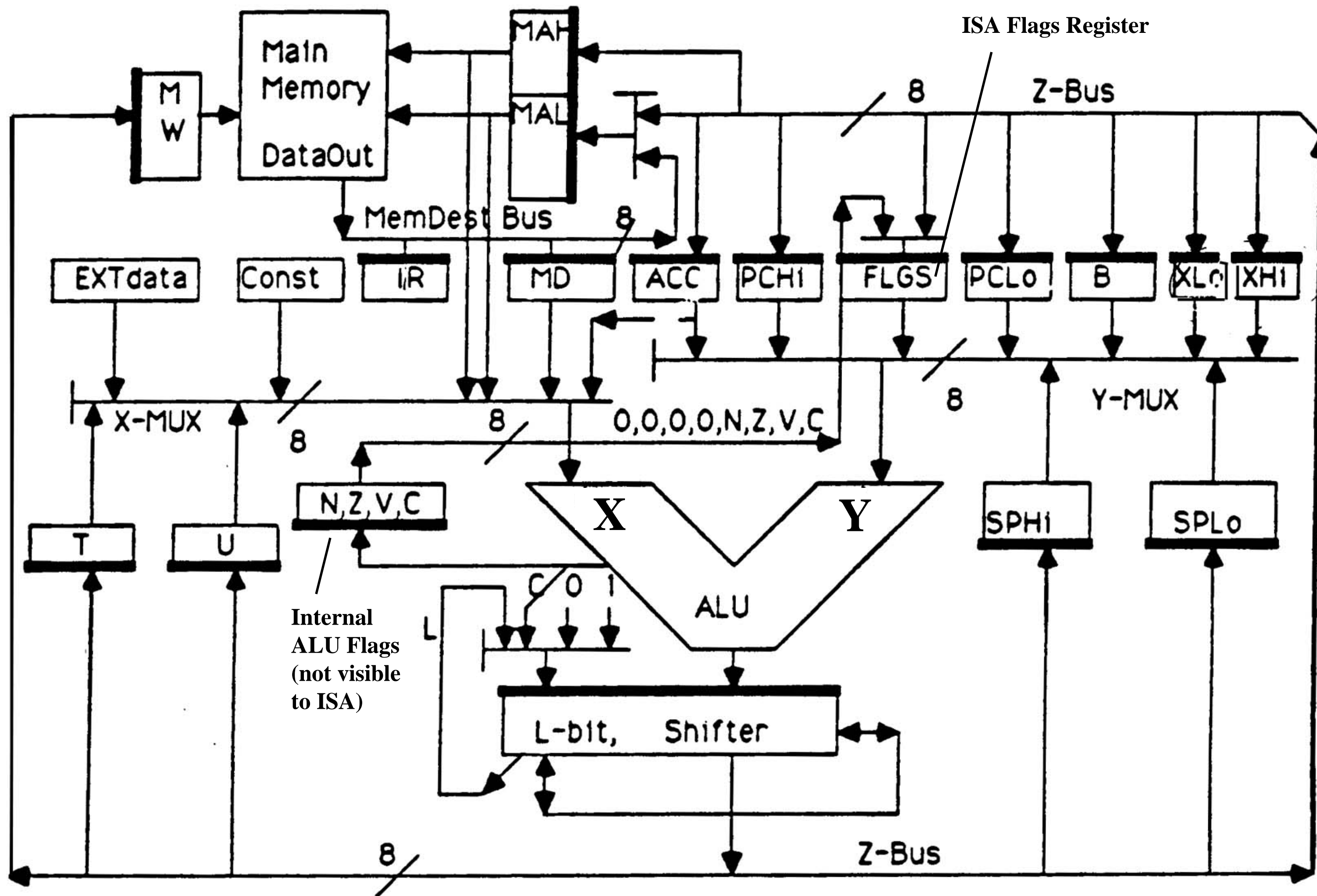
Independent RTN:

Instruction ← Mem[PC]
AddressHi ← Mem[PC + 1]
AddressLo ← Mem[PC + 2]
PC ← PC + 3
Accumulator ← Mem[Mem[Address]]

PC = PCHi,PCLo
Address = AddressHi,Address Lo

16 bit memory address – memory width one byte

The Datapath



Datapath Details

- **ISA Registers:**
  - **Program Counter (PC) 16 bits implemented in datapath by two 8-bit registers: PCHi, PCLo**
  - **Accumulator (ACC) 8-bit register.**
  - **Flags Register (FLGS) 8-bit register, low four bits are the flags: NZVC**
  - **Stack Pointer (SP) 16-bit register implemented in datapath by two 8-bit registers: SPHi, SPLo**
- **Memory Details**
  - **A single main memory used for instructions and data. 16-bit address.**
  - **Memory Address High Register (MAH) must be loaded first for both reads and writes.**
  - **For memory writes the MW (Memory Write) register must be loaded with the byte to be stored.**
  - **Reading from and writing to memory is triggered by loading MAL (Memory Address Low) register using ALUDEST microinstruction field options.**
  - **Memory read/write operations take <u>two clock cycles</u> from the start of a memory read or write.**
- **Temporary Datapath/Microprogram Registers:**
  - **The following 8 -bit registers can be used by the control microprogram to store temporary values as needed and are not visible to the ISA or user programs: T, U, B, XLo, Xhi**
- **The ALU Output Shifter:**
  - **Combinational logic shifter that can shift the ALU output one position left or right and also manipulate the most significant bit of the ALU output (L-bit) before shifting in a single cycle.**
- **Constant Value (Const):**
  - **Shown as a possible input to the ALU is 5-bit value that can be specified by a microinstruction field (field H).**

The Microinstruction Format

Total 14 Fields

A	B	C	D	E	F	G	H	I	J	K	L	M	N
MEMDEST	LCNTRL	SHFTCTRL	ALUCTRL	YSOURCE	XSOURCE	ALUDEST	CONST	LOADFLGS	TEST	INTERNAB	ADDRF	COND	OPCODE
Bits 0-1 (2 bits)	Bits 2-3 (2 bits)	Bits 4-5 (2 bits)	Bits 6-9 (4 bits)	Bits 10-13 (4 bits)	Bits 14-16 (3 bits)	Bits 17-20 (4 bits)	Bits 21-25 (5 bits)	Bit 26 (1 bit)	Bits 27-28 (2 bits)	Bit 29 (1 bit)	Bits 30-38 (9 bits)	Bits 39-40 (2 bits)	Bits 41-42 (2 bits)

Bit	Field	Name	Operations	
0, 1	A	MEMDEST	00 (0) - NOP (See Note 2) 01 (1) - MD 10 (2) - MD and MALow 11 (3) - MD and MALow and IR	
2, 3	B	LCNTRL	00 (0) - Leave L alone 01 (1) - Clear L 10 (2) - Set L 11 (3) - L = Carry Out of ALU	
4, 5	C	SHFTNTRL	00 (0) - No Shift 01 (1) - Shift Right 10 (2) - Shift Left 11 (3) - Not Used	
6, 7, 8, 9	D	ALUCTRL	0000 (0) - X 0001 (1) - Y 0010 (2) - X plus Y 0011 (3) - X plus Y plus 1 0100 (4) - X and Y 0101 (5) - X or Y 0110 (6) - X xor Y 0111 (7) - not Y	1000 (8) - X plus 1 1001 (9) - Y plus 1 1010 (10) - X and 1 1011 (11) - Y and 1 1100 (12) - Y plus not X plus 1 1101 (13) - not X 1110 (14) - minus 1 (i.e. the hex value FF) 1111 (15) - 0
10, 11, 12, 13	E	YSOURCE	0000 (0) - none 0001 (1) - ACC 0010 (2) - PCLo 0011 (3) - SPLo 0100 (4) - B 0101 (5) - FLAGS 0110 (6) - XHi 0111 (7) - XLo	1000 (8) - PCHi 1001 (9) - SPHi 1010 (10) - unused 1011 (11) - unused 1100 (12) - unused 1101 (13) - unused 1110 (14) - unused 1111 (15) - unused
14, 15, 16	F	XSOURCE	000 (0) - ACC 001 (1) - MD 010 (2) - CONST (Constant Field from Microinstruction) 011 (3) - External Data (not used here) 100 (4) - T 101 (5) - MALo 110 (6) - MAHi 111 (7) - U	

Field values above = Binary (Decimal) - Functionality
All microinstruction field values in cmemory file must in decimal

The Microinstruction Format

Total 14 Fields

A	B	C	D	E	F	G	H	I	J	K	L	M	N
MEMDEST	LCNTRL	SHFTCTRL	ALUCTRL	YSOURCE	XSOURCE	ALUDEST	CONST	LOADFLGS	TEST	INTERNAB	ADDRF	COND	OPCODE
Bits 0-1	Bits 2-3	Bits 4-5	Bits 6-9	Bits 10-13	Bits 14-16	Bits 17-20	Bits 21-25	Bit 26	Bits 27-28	Bit 29	Bits 30-38	Bits 39-40	Bits 41-42
(2 bits)	(2 bits)	(2 bits)	(4 bits)	(4 bits)	(3 bits)	(4 bits)	(5 bits)	(1 bit)	(2 bits)	(1 bit)	(9 bits)	(2 bits)	(2 bits)

Bit	Field	Name	Operations	
17 18 19 20	G	ALUDEST	0000 (0) - none 0001 (1) - ACC 0010 (2) - PCLo 0011 (3) - SPLo 0100 (4) - B 0101 (5) - FLAGS 0110 (6) - XHi 0111 (7) - XLo	1000 (8) - PCHi 1001 (9) - SPHi 1010 (10) - MAHi 1011 (11) - MALo, Read (Starts Memory Read) 1100 (12) - T 1101 (13) - MALo, Write (Starts Memory Write) 1110 (14) - U 1111 (15) - MW
21 22 23 24 25	H	CONST	Unsigned 5-bit constant for XSOURCE Value Range: 00000-11111 (0-31)	
26	I	LOADFLGS	When 1 loads FLAGS from internal ALU flags NBIT, ZBIT, VBIT, CBIT	
27 28	J	TEST	00 (0) - Branch on NBIT 01 (1) - Branch on ZBIT 10 (2) - Branch on VBIT 11 (3) - Branch on CBIT	(See Note 1)
29	K	INTRENAB	Not Used, always put 0 in this field	
30 31 32 33 34 35 36 36 38	L	ADDRF	9-bit address field of Next microinstruction bit 30: Most Significant Bit of Address (See Note 1) Address range: 0-511 (Decimal)	
39 40	M	COND	Determines Type of Next Microinstruction Address (See note 1)	
41 42	N	OPCODE	Opcode: Format of Microinstruction Only one format used here, always put 0 in this field.	

Field values above = Binary (Decimal) - Functionality
All microinstruction field values in cmemory file must in decimal

Microinstruction Fields Notes

A	B	C	D	E	F	G	H	I	J	K	L	M	N
MEMDEST Bits 0-1 (2 bits)	LCNTRL Bits 2-3 (2 bits)	SHFTCTRL Bits 4-5 (2 bits)	ALUCTRL Bits 6-9 (4 bits)	YSOURCE Bits 10-13 (4 bits)	XSOURCE Bits 14-16 (3 bits)	ALUDEST Bits 17-20 (4 bits)	CONST Bits 21-25 (5 bits)	LOADFLGS Bit 26 (1 bit)	TEST Bits 27-28 (2 bits)	INTERNAB Bit 29 (1 bit)	ADDRF Bits 30-38 (9 bits)	COND Bits 39-40 (2 bits)	OPCODE Bits 41-42 (2 bits)

Note 1:

If COND = 00 (0 decimal) then MPC (next microinstruction address) = ADDRF (i.e bits 30-38) as given

If COND = 01 (1 decimal) Then MPC (next microinstruction address) is determined by bits 30-37 of ADDRF along with the particular test bit specified by TEST field from the ALU replacing the least significant bit of 38 of ADDRF (i.e two way branch on the condition bit tested).

If COND = 10 (2 decimal) Then MPC (next microinstruction address) by bits 30-34 (five most significant bits of ADDRF) along with the 4 most significant bits of IR (instruction register) replacing the low 4 bits 35-38 of ADDRF (I.e 16-way branch on the 4 most significant bits of IR).

If COND = 11 (3 decimal) Then MPC (next microinstruction address) by bits 30-34 (five most significant bits of ADDRF) along with the 4 least significant bits of IR (instruction register) replacing the low 4 bits 35-38 of ADDRF (I.e 16-way branch on the 4 least significant bits of IR).

Note 2: The Memory destination from the memory bus MBUS (memory data bus) is as follows:

When the MEMDEST field is not 00 (0 decimal), MD is loaded from MBUS.

When the field is 10 (2 decimal), MD and MALo registers are loaded from MBUS

When the field is 11 (3 decimal) MD, MALo and IR registers are loaded from MBUS

Note 3: For every microinstruction field, use the decimal value of the binary field values specified above, as shown in the sample microprogram start segment on the next page.

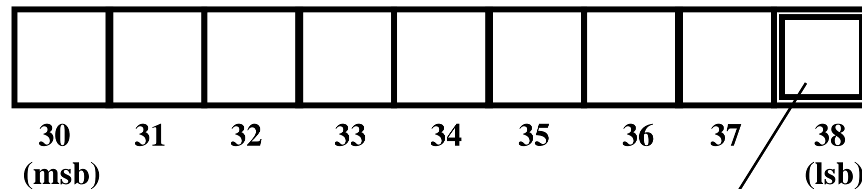
Microinstruction 2-Way Branch

COND = 1

<u>When microinstruction COND field is set to 01 (1 decimal):</u>

Then next microinstruction address is determined by bits 30-37 of the address field ADDRF along with the particular test bit (internal ALU flag) specified by TEST field from the ALU replacing the least significant bit 38 of ADDRF (i.e two way branch on the condition bit tested).

Address Field ADDRF



TEST = 00 (0) - Branch on NBIT

TEST = 01 (1) - Branch on ZBIT

TEST = 10 (2) - Branch on VBIT

TEST = 11 (3) - Branch on CBIT

Least significant address bit replaced with internal ALU Flag specified by TEST field

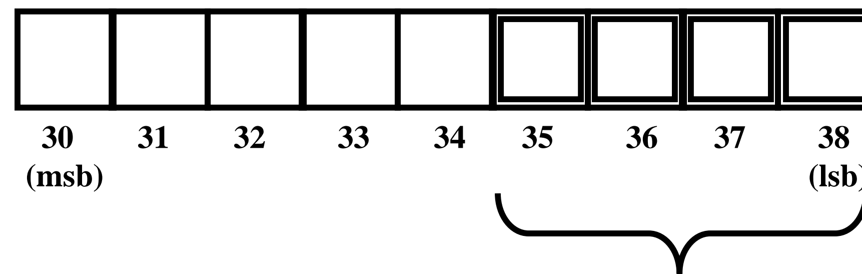
Microinstruction 16-Way Branch On Upper-Half of Opcode

COND = 2

When microinstruction COND field is set to 10 (2 decimal):

Then next microinstruction address is determined by bits 30-34 (five most significant bits of ADDRF) along with the 4 most significant bits of Opcode in IR (instruction register) replacing the low 4 bits 35-38 of ADDRF (i.e 16-way branch on the 4 most significant bits of opcode in IR).

Address Field ADDRF



4 least significant address bits (35-38) of ADDRF replaced with 4 most significant bits of Opcode in IR

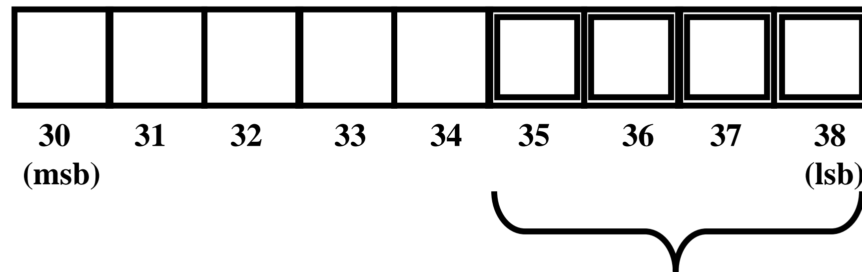
Microinstruction 16-Way Branch On Lower-Half of Opcode

COND = 3

<u>When microinstruction COND field is set to 11 (3 decimal):</u>

Then next microinstruction address is determined by bits 30-34 (five most significant bits of ADDRF) along with the 4 least significant bits of Opcode in IR (instruction register) replacing the low 4 bits 35-38 of ADDRF (i.e 16-way branch on the 4 least significant bits of opcode in IR).

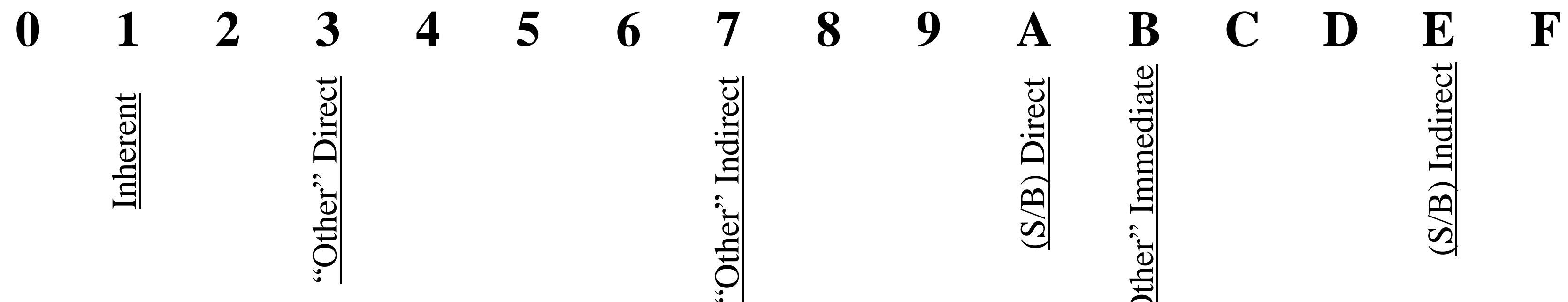
Address Field ADDRF



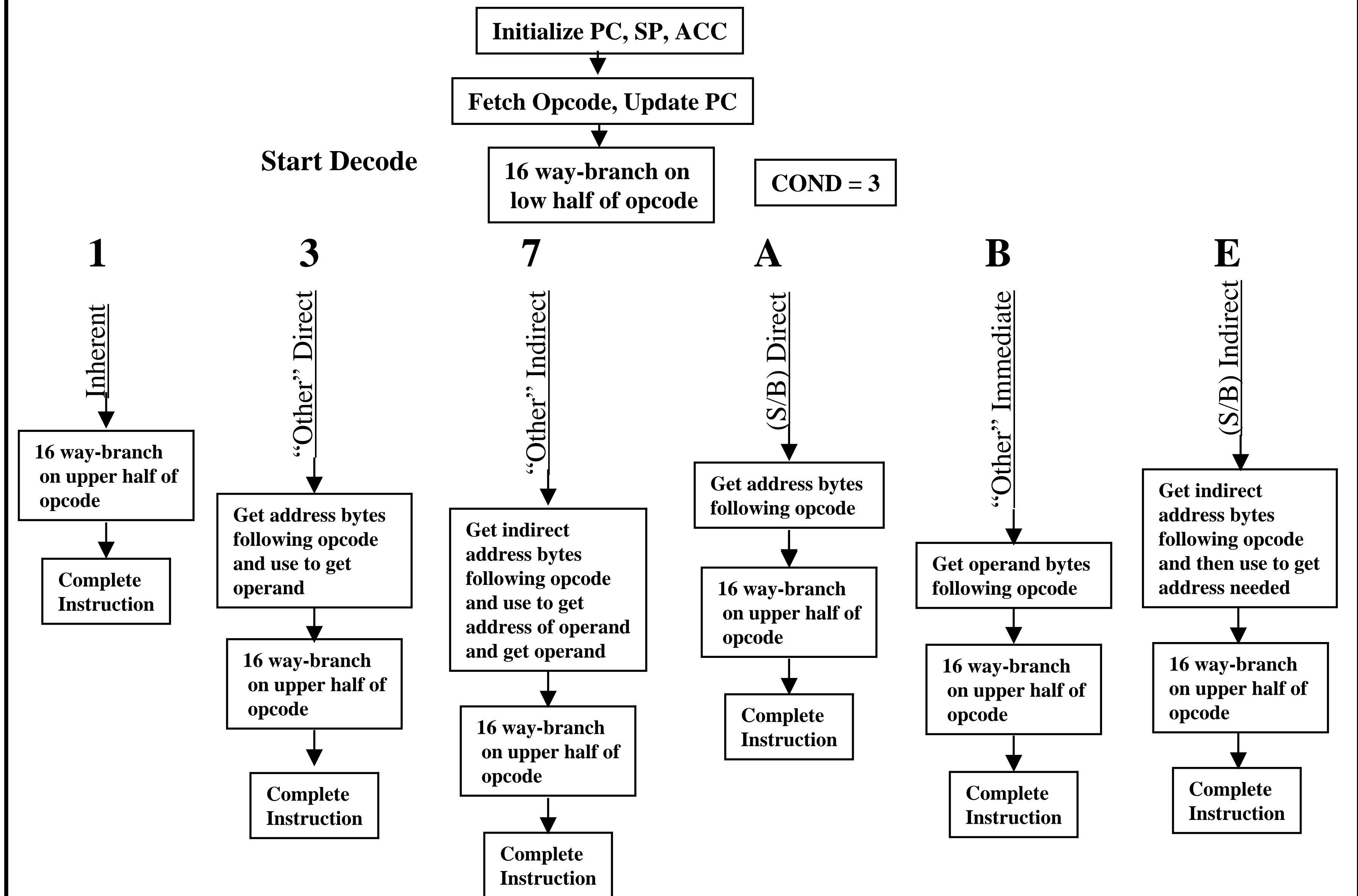
4 least significant address bits (35-38) of ADDRF replaced with 4 least significant bits of Opcode in IR

Six Types of Instructions According to Low-Half of Opcode

- **<u>Inherent Instructions:</u> Low half of Opcode = 0001 = 1 (hex)**
  - **One Byte: Opcode**
- **<u>Store/Branch (S/B) Direct:</u> Low half of Opcode = 1010 = a (hex)**
  - **Opcode, two direct address bytes following opcode**
- **<u>Store/Branch (S/B) Indirect:</u> Low half of Opcode = 11010 = e (hex)**
  - **Opcode, two indirect address bytes following opcode**
  - **Need to get two more bytes of address**
- **<u>"Other" Direct:</u> Low half of Opcode = 0011 = 3 (hex)**
  - **Opcode, two bytes of direct address of operand following opcode**
  - **Direct address is then used to fetch operand byte (or two bytes in case of LDS)**
- **<u>"Other" Indirect:</u> Low half of Opcode = 0111 = 7 (hex)**
  - **Opcode, two indirect bytes of address following opcode**
  - **Need to get two more bytes of address of operand**
  - **Operand address is then used to fetch operand byte (or two bytes in case of LDS)**
- **<u>"Other" Immediate:</u> Low half of Opcode = 1011 = b (hex)**
  - **Opcode, followed with one byte of operand (or two bytes in case of LDS)**



A Possible High-Level Microprogram Flow



Sample Microprogram Start Segment

From project zip file: start-microprogram.mc

Includes: initialization Opcode fetch, partial decode

```
# Sample microprogram start segment
#
#  A - MemDest
# | B - LCtrl
# | | C - ShftCtrl
# | | | D - AluCtrl
# | | | | E - YSrc
# | | | | | F - XSrc
# | | | | | | G - AluDest
# | | | | | | | H - Const
# | | | | | | | | I - LdFlg
# | | | | | | | | | J - Test
# | | | | | | | | | | K - Intrne
# | | | | | | | | | | | L - Addr
# | | | | | | | | | | | | M - Cond
# | | | | | | | | | | | | | N - OpCode
# | | | | | | | | | | | | | |; addr: comment

# Initialize
# 000: PcHi  ←  0
memdest=0,lctrl=0,shftctrl=0,aluctrl=15,ysource=0,xsource=0,aludest=8,const=0,loadflgs=0,test=0,intrenab=0,addrf=1,cond=0,opcode=0;
# 001: PcLo ← 0
memdest=0,lctrl=0,shftctrl=0,aluctrl=15,ysource=0,xsource=0,aludest=2,const=0,loadflgs=0,test=0,intrenab=0,addrf=2,cond=0,opcode=0;
# 002: SpHi  ← 255
memdest=0,lctrl=0,shftctrl=0,aluctrl=14,ysource=0,xsource=0,aludest=9,const=0,loadflgs=0,test=0,intrenab=0,addrf=3,cond=0,opcode=0;
# 003: SpLo ← 255
memdest=0,lctrl=0,shftctrl=0,aluctrl=14,ysource=0,xsource=0,aludest=3,const=0,loadflgs=0,test=0,intrenab=0,addrf=4,cond=0,opcode=0;
# 004: ACC ← 0
memdest=0,lctrl=0,shftctrl=0,aluctrl=15,ysource=0,xsource=0,aludest=1,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
```

Sample Microprogram Start Segment (Continued)

```
# Fetch
# 005: MaHi ← PcHi - isa  label used to calculate CPI
isa:
memdest=0,lctrl=0,shftctrl=0,aluctrl=1,ysource=8,xsource=0,aludest=10,const=0,loadflgs=0,test=0,intrenab=0,addrf=6,cond=0,opcode=0;
# 006: MaLo ← PcLo, MemRead (start fetch)
memdest=0,lctrl=0,shftctrl=0,aluctrl=1,ysource=2,xsource=0,aludest=11,const=0,loadflgs=0,test=0,intrenab=0,addrf=7,cond=0,opcode=0;
# 007: PcLo += 1  (also wait cycle)
memdest=0,lctrl=0,shftctrl=0,aluctrl=9,ysource=2,xsource=0,aludest=2,const=0,loadflgs=0,test=3,intrenab=0,addrf=8,cond=1,opcode=0;
# 008: C is 0,   IR ← Mem(PC)
memdest=3,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=10,cond=0,opcode=0;
# 009: C is 1, IR ← Mem(PC),  PcHi += 1
memdest=3,lctrl=0,shftctrl=0,aluctrl=9,ysource=8,xsource=0,aludest=8,const=0,loadflgs=0,test=0,intrenab=0,addrf=10,cond=0,opcode=0;

#  END OPCODE FETCH, START DECODE
```

Sample Microprogram Start Segment (Continued)

```
# 010:  Start decode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=16,cond=3,opcode=0;
# 011: No-OP Filler
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=16,cond=0,opcode=0;
# 012: No-OP Filler
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=16,cond=0,opcode=0;
# 013: No-OP Filler
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=16,cond=0,opcode=0;
# 014: No-OP Filler
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=16,cond=0,opcode=0;
# 015: No-OP Filler
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=16,cond=0,opcode=0;

#;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

# Begin first stage decode - 16-way branch on lower half of opcode byte
# 016: 0 NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 017: 1 Inherent
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 018: 2 NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 019: 3 Other: Direct Addressing
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 020: 4 NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 021: 5 NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 029: 6 NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=2,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 023: 7 Other: Indirect Addressing
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 024: 8 NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 025: 9 NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=2,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 026: A Store and Branch: Direct Addressing
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=2,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 027: B Other: Immediate Addressing
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=2,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 028: C NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 029: D NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=2,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 030: E Store and Branch: Indirect Addressing
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 031: F NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;

#;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```